# Projeto e Análise de Algoritmos

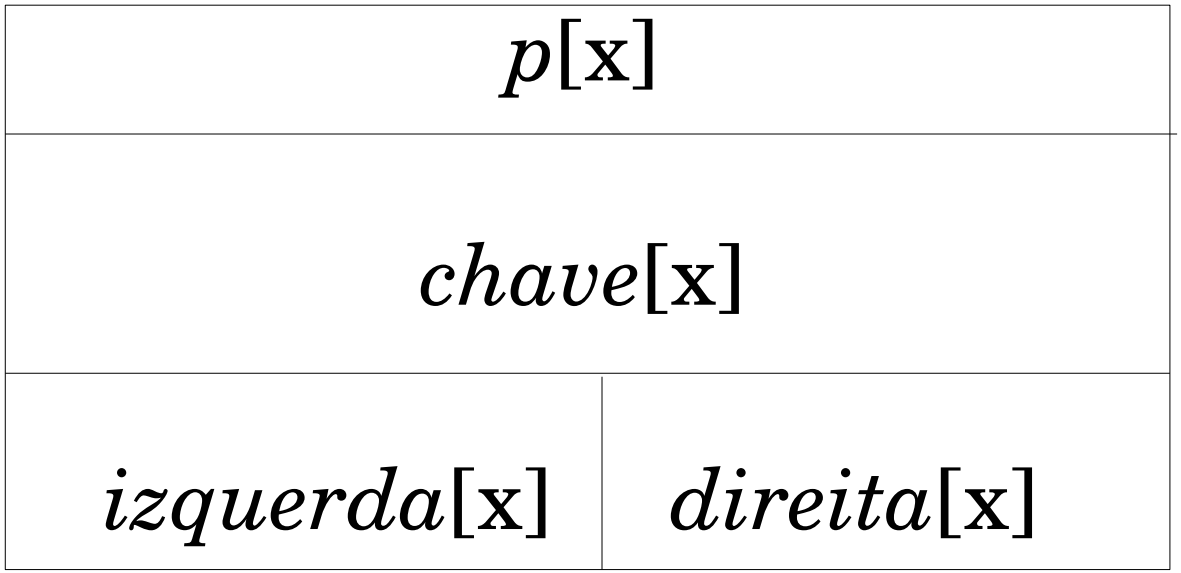
Árvores de pesquisa binária
Aula 12

Diane Castonguay
diane@inf.ufg.br

Instituto de Informática
Universidade Federal de Goiás

# Árvores de pesquisa binária

objeto x




sub-arvore com raiz *izquerda*[x] ≤ *chave*[x]

sub-arvore com raiz *direita*[x] ≥ *chave*[x]

# Ordenar

```
INORDER-TREE-WALK(x)
1. se x ≠ NIL
2.     então //arvore esquerda tem valores ≤ chave[x]
3.             INORDER-TREE-WALK(esquerda[x])
4.             imprima chave[x]
5.           //arvore direita tem valores ≥ chave[x]
6.             INORDER-TREE-WALK(direita[x])
7. fim-se
```

# Pesquisar - Recursivamente

```
TREE-SEARCH(x, k)
1. se x = NIL ou k = chave[x]
2.       então retorna x
3.  fim-se
4. se k < chave[x]
5.       então retorna TREE-SEARCH(esquerda[x], k)
6.       senão retorna TREE-SEARCH(direita[x], k)
7.  fim-se
```

# Pesquisar - Iterativamente

```
ITERATIVE-TREE-SEARCH(x, k)
1. enquanto x ≠ NIL e k ≠ chave[x] faça
2.       se k < chave[x]
3.             então x ← esquerda[x]
4.             senão x ← direita[x]
5.       fim-se
6. fim-enquanto
7. retorna x
```

# Mínimo e máximo

TREE-MINIMUM(x)

1. enquanto *esquerda*[x] ≠ NIL faça
2. x ← *esquerda*[x]
3. fim-enquanto
4. retorna x

TREE-MAXIMUM(x)

1. enquanto *direita*[x] ≠ NIL faça
2. x ← *direita*[x]
3. fim-enquanto
4. retorna x

# Sucessor

TREE-SUCESSOR(x)
1. se *direita*[x] ≠ NIL
2. então retorna TREE-MINIMUM(*direita*[x])
3. fim-se
4. $y \leftarrow p$[x]
5. enquanto $y \neq$ NIL e x = *direita*[$y$] faça
6. x $\leftarrow y$
7. $y \leftarrow p$[x]
8. fim-enquanto
9. retorna $y$

# Inserção - 1

TREE-INSERT(T, z)

1. $y \leftarrow$ NIL
2. $x \leftarrow raiz$[T]
3. enquanto $x \neq$ NIL faça
4. $\quad y \leftarrow x$
5. $\quad$ se $chave$[z] < $chave$[$x$]
6. $\quad\quad$ então $x \leftarrow esquerda$[$x$]
7. $\quad\quad$ senão $x \leftarrow direita$[$x$]
8. $\quad$ fim-se
9. fim-enquanto

# Inserção - 2

*TREE-INSERT(T, z) (continação)*

10. $p$[z] ← $y$
11. se $y$ = NIL
12. então *raiz*[T] ← z
13. senão se *chave*[z] < *chave*[$x$]
14. então *esquerda*[$y$] ← z
15. senão *direita*[$y$] ← z
16. fim-se
17. fim-se

# Eliminação - 1

TREE-DELETE(T, z)

1. se *esquerda*[z] = NIL ou *direita*[z] = NIL
2. então $y \leftarrow$ z
3. senão $y \leftarrow$ TREE-SUCESSOR(z)
4. fim-se

// $y$ possui ao máximo um filho

5. se *esquerda*[$y$] $\neq$ NIL
6. então $x \leftarrow$ *esquerda*[$y$]
7. senão $x \leftarrow$ *direita*[$y$]
8. fim-se

// $x$ é o filho de $y$. Se for da direita, ele é o único.

# Eliminação - 2

*TREE-DELETE(T, z) (continuação)*

// tirar o nó $y$ e colcar $x$ no lugar dele

```
9.   se x ≠ NIL
10.        então p[x] ← p[y]
11. fim-se
12. se p[y] = NIL
13.    então raiz[T] ← x
14.    senão    se y = esquerda[p[y]]
15.                 então esquerda[p[y]] ← x
16.                 senão direita[p[y]] ← x
17.              fim-se
18. fim-se
```

# Eliminação - 3

*TREE-DELETE(T, z) (continuação)*

//copiar as informações de y no lugar do z e
//liberar o nó y

19. se $y \neq$ z
20. então *chave*[z] ← *chave*[*y*]
21. copiar dados satélite de *y* em z
22. fim-se
23. retorna *y*